Potential Problems in Using JLex

The following differences from "standard" Lex notation appear in JLex:

- Escaped characters within quoted strings are not recognized. Hence **`"\n"`** is *not* a new line character. Escaped characters outside of quoted strings (**`\n`**) and escaped characters within character classes (**`[\n]`**) are OK.
- A blank should not be used within a character class (i.e., **`[`** and **`]`**). You may use **`\040`** (which is the character code for a blank).

- A doublequote must be escaped within a character class. Use **`[\"]`** instead of **`["]`**.
- White space is defined to be all characters before blank as well as the last ASCII character. These can be represented as: **`[\000-\037\177]`**

JLex Examples

A JLex scanner that looks for five letter words that begin with “P” and end with “T”.

This example is in
`~cs536-1/public/jlex`

The JLex specification file is:

```
class Token {
   String text;
   Token(String t){text = t;}
}
%%
Digit=[0-9]
AnyLet=[A-Za-z]
Others=[0-9’&.]
WhiteSp=[\040\n]
// Tell JLex to have yylex() return a
Token
%type Token
// Tell JLex what to return when eof of
file is hit
%eofval{
return new Token(null);
%eofval}
%%
[Pp]{AnyLet}{AnyLet}{AnyLet}[Tt]{WhiteSp}+
     {return new Token(yytext());}

({AnyLet}|{Others})+{WhiteSp}+
     {/*skip*/}
```

The Java program that uses the scanner is:

```
import java.io.*;

class Main {

public static void main(String args[])
   throws java.io.IOException {

 Yylex lex  = new Yylex(System.in);
 Token token = lex.yylex();

 while ( token.text != null ) {
   System.out.print("\t"+token.text);
   token = lex.yylex(); //get next token
 }
}}
```

In case you care, the words that are matched include:

```
Pabst
paint
petit
pilot
pivot
plant
pleat
point
posit
Pratt
print
```

A JLex tester that looks for matches of regular expressions being tested.

This example is in
`~cs536-1/public/jlex.tester`

The JLex specification file is:

```
class Token {
   String text;
   Token(String t){text = t;}
}
%%
Digit=[0-9]
AnyLet=[A-Za-z]
Others=[0-9’&.]
WhiteSp=[\040\n]
// Tell JLex to have yylex() return a
Token
%type Token
// Tell JLex what to return when eof of
file is hit
%eofval{
return new Token(null);
%eofval}
%%
testRE      {return new Token(yytext());}
{WhiteSp}+ {/*skip*/}
(.)         {System.out.println(
              "Illegal:"+yytext());}
```

The Java program that uses this scanner tester is:

```
import java.io.*;

class Main {

public static void main(String args[])
   throws java.io.IOException {

 Yylex lex  = new Yylex(System.in);
 Token token = lex.yylex();

 while ( token.text != null ) {
   System.out.print("Matched:"+
        token.text);
   token = lex.yylex(); //get next token
 }
}}
```

An example of CSX token specifications. This example is in
`~cs536-1/public/proj2/startup`

The JLex specification file is:

```
import java_cup.runtime.*;

/*  Expand this into your solution for
project 2 */

class CSXToken {
 int linenum;
 int colnum;
 CSXToken(int line,int col){
 linenum=line;colnum=col;};
}

class CSXIntLitToken extends CSXToken {
 int intValue;
 CSXIntLitToken(int val,int line,
   int col){
   super(line,col);intValue=val;};
}

class CSXIdentifierToken extends
CSXToken {
String identifierText;
CSXIdentifierToken(String text,int line,
  int col){
  super(line,col);identifierText=text;};
}
```

```
class CSXCharLitToken extends CSXToken {
  char charValue;
CSXCharLitToken(char val,int line,
    int col){
   super(line,col);charValue=val;};
}

class CSXStringLitToken extends CSXToken
{
   String stringText;
   CSXStringLitToken(String text,
     int line,int col){
   super(line,col);
   stringText=text; };
}

// This class is used to track line and
column numbers
// Feel free to change to extend it
class Pos {
static int  linenum = 1;
/* maintain this as line number current
   token was scanned on */
static int  colnum = 1;
  /* maintain this as column number
    current token began at */
static int  line = 1;
/* maintain this as line number after
   scanning current token  */
static int  col = 1;
  /* maintain this as column number
    after scanning current token  */
static void setpos() {
 //set starting pos for current token
   linenum = line;
   colnum = col;}
}

%%
Digit=[0-9]

// Tell JLex to have yylex() return a
 Symbol, as JavaCUP will require
%type Symbol

// Tell JLex what to return when eof of
file is hit
%eofval{
return new Symbol(sym.EOF,
         new  CSXToken(0,0));
%eofval}

%%
"+"   {Pos.setpos(); Pos.col +=1;
        return new Symbol(sym.PLUS,
          new CSXToken(Pos.linenum,
                     Pos.colnum));}
```

```
"!="     {Pos.setpos(); Pos.col +=2;
          return new Symbol(sym.NOTEQ,
            new CSXToken(Pos.linenum,
                         Pos.colnum));}
";"      {Pos.setpos(); Pos.col +=1;
            return new Symbol(sym.SEMI,
              new CSXToken(Pos.linenum,
                           Pos.colnum));}
{Digit}+    {// This def doesn't check
             // for overflow
            Pos.setpos();
            Pos.col += yytext().length();
            return new Symbol(sym.INTLIT,
             new CSXIntLitToken(
        new Integer(yytext()).intValue(),
        Pos.linenum,Pos.colnum));}

\n      {Pos.line +=1; Pos.col = 1;}
" "     {Pos.col +=1;}
```

The Java program that uses this scanner (P2) is:

```
class P2 {
 public static void main(String args[])
    throws java.io.IOException {
  if (args.length != 1) {
    System.out.println(
    "Error: Input file must be named on
command line." );
    System.exit(-1);
  }
  java.io.FileInputStream yyin = null;
  try {
   yyin =
   new java.io.FileInputStream(args[0]);
  } catch (FileNotFoundException
           notFound){
    System.out.println(
   "Error: unable to open input file.");
    System.exit(-1);
  }

 // lex is a JLex-generated scanner that
 // will read from yyin
    Yylex lex = new Yylex(yyin);
    System.out.println(
       "Begin test of CSX scanner.");

/**********************************
 You should enter code here that
 thoroughly test your scanner.

 Be sure to test extreme cases,
 like very long symbols or lines,
 illegal tokens, unrepresentable
 integers, illegals strings, etc.
 The following is only a starting point.
***********************************/
Symbol token = lex.yylex();

while ( token.sym != sym.EOF ) {
 System.out.print(
   ((CSXToken) token.value).linenum
   + ":"
   + ((CSXToken) token.value).colnum
   + " ");

  switch (token.sym) {
   case sym.INTLIT:
    System.out.println(
     "\tinteger literal(" +
     ((CSXIntLitToken)
     token.value).intValue + ")");
    break;
  case sym.PLUS:
    System.out.println("\t+");
    break;

  case sym.NOTEQ:
    System.out.println("\t!=");
    break;

  default:
    throw new RuntimeException();
 }

 token = lex.yylex(); // get next token
}

System.out.println(
   "End test of CSX scanner.");
}}}
```